External Modeling Framework and the OpenUTF[1]

Jeffrey S. Steinman, Ph.D.
Craig N. Lammers
Maria E. Valinski
Wendy L. Steinman

WarpIV Technologies, Inc.
5230 Carroll Canyon Road, Suite 306
San Diego, CA 92121

(858) 605-1646

steinman@warpiv.com, craig.lammers@warpiv.com, maria.valinski@warpiv.com, wendy.steinman@warpiv.com

Key Words:
External Modeling Framework, EMF, Forces Modeling and Simulation, Open Unified Technical Framework, Parallel and Distributed Modeling and Simulation, High Performance Computing, Multicore, Manycore, WarpIV Kernel

ABSTRACT: *The Open Unified Technical Framework (OpenUTF) provides an External Modeling Framework (EMF) that offers support for all of its standard modeling constructs within an encapsulated object that can be created and used by non-OpenUTF applications. Acting like a proxy to simulations executing in the OpenUTF, the EMF coordinates robust and repeatable event processing and state management in logical time between external applications and the core parallel and distributed OpenUTF simulation. Like the High Level Architecture (HLA), applications can optionally use their own simulation engines to coordinate their internal event processing with OpenUTF simulations using the EMF. Applications can also integrate with standard OpenUTF-compliant models that execute directly within the EMF. Because the EMF and HLA provide similar functionality, it is straightforward to implement an HLA interface as a wrapper for the EMF to facilitate direct HLA interoperability between external systems and simulations executing within the OpenUTF.*

All events within the EMF are processed conservatively in logical time. Yet, the EMF supports rollbackable and rollforwardable state management as events are internally processed, not for the purpose of handling straggler messages to maintain causality, but for performing state reconstruction at any logical time in the past, present, or future. The EMF provides publish and subscribe services along with standard methods to access remote objects and their attributes. One very common use of the EMF is to support live or offline visualization and analysis capabilities for OpenUTF simulations. With a VCR-like set of controls, graphics-based applications can freely move forward and backward to any simulated time in the past or present to visualize the state of the simulation while it is executing. All received messages are time tagged and can optionally be logged for playback when running live. This facilitates powerful after-action-review visualization and analysis with complete state reconstruction at any point in time. The EMF provides the foundation for supporting analysis, command and control, integration with real systems, warfighter training, test & evaluation, and formal verification validation & accreditation use cases.

This paper first provides historical background on previous EMF implementations that were developed for the Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES), leading to its current implementation within the WarpIV Kernel. The paper then describes the current set of EMF services that are provided for the OpenUTF, its high-level design, and its recent implementation. This topic is important to the Simulation Interoperability Standards Organization (SISO) because the EMF is a core architectural component of the OpenUTF that is being investigated by the Parallel and Distributed Modeling & Simulation Standing Study Group (PDMS-SSG) for future standardization.

[1] Approved for public release, 12-MDA-6532 (24 January 12).

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **24 JAN 2012**	2. REPORT TYPE	3. DATES COVERED **00-00-2012 to 00-00-2012**
4. TITLE AND SUBTITLE **External Modeling Framework And The OpenUTF1**		5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **WarpIV Technologies, Inc,5230 Carroll Canyon Road, Suite 306,San Diego,CA,92121**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S) 11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Open Unified Technical Framework (OpenUTF) provides an External Modeling Framework (EMF) that offers support for all of its standard modeling constructs within an encapsulated object that can be created and used by non-OpenUTF applications. Acting like a proxy to simulations executing in the OpenUTF, the EMF coordinates robust and repeatable event processing and state management in logical time between external applications and the core parallel and distributed OpenUTF simulation. Like the High Level Architecture (HLA),applications can optionally use their own simulation engines to coordinate their internal event processing with OpenUTF simulations using the EMF. Applications can also integrate with standard OpenUTF-compliant models that execute directly within the EMF. Because the EMF and HLA provide similar functionality, it is straightforward to implement an HLA interface as a wrapper for the EMF to facilitate direct HLA interoperability between external systems and simulations executing within the OpenUTF.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**11**	

1.0 Introduction

The External Modeling Framework (EMF) has undergone several generations of innovation, design, and development, leading to the current capability that resides within the WarpIV Kernel implementation of the Open Unified Technical Framework (OpenUTF). The EMF provides a technical foundation for interfacing remote systems that are potentially distributed across wide area networks with an optimistic simulation executing on parallel and distributed computers. Examples of such remote systems might include (a) visualization and analysis tools, (b) hardware-in-the-loop, (c) simulations executing on remote machines, (d) real-world software systems and applications, (e) training systems, and/or (f) distributed test articles being verified and validated within a robust and scalable OpenUTF-based testbed for correct operation.

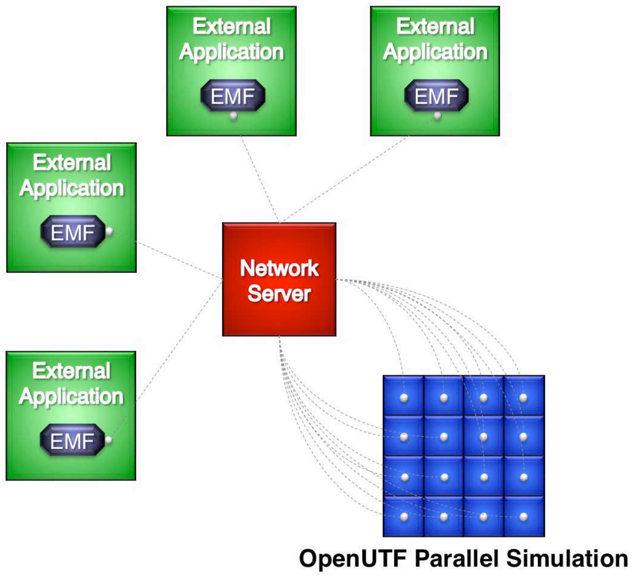

Figure 1: EMF in the OpenUTF. In this example a 16-node OpenUTF simulation executes in parallel. Each node has a Simulation Client that connects through a TCP/IP Socket to the Network Server. In addition, each External Application has an EMF and an External Client that also connects to the Network Server. All connections to the Network Server are two-way, which coordinates message traffic between External Applications and the OpenUTF Parallel Simulation. Note that External Applications never communicate directly with one another. Message traffic includes initialization data, time requests and advances, two-way event scheduling, Federation Object discovery/removal and attribute updates.

Figure 1 depicts the conceptual operation of the EMF operating as the remote interface for external applications as they interact with an OpenUTF parallel simulation. Note that any number of external applications can be supported with dynamic connectivity. External applications can reside wherever network connectivity is provided.[2]

This rest of this section describes the evolution of each EMF implementation within its historical context.

1990-1993: Jet Propulsion Laboratory and CalTech

The first implementation of the EMF[3] was developed at the Jet Propulsion Laboratory (JPL) between 1990 and 1993. This early implementation was groundbreaking in several ways. It introduced all of the core technologies for synchronizing external systems such as graphical visualization tools and hardware-in-the-loop test articles with optimistic simulations executing on parallel computers. While primitive in several ways,[4] this first implementation pioneered the robust technical foundation for supporting remote visualization of parallel and distributed Missile and Air Defense simulation applications executing within the Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES) on supercomputers.[5]

This early EMF abstracted (a) all of the communications between external visualization tools and other remote applications with the simulation, (b) coordinated two-way event processing in logical time between remote applications and the parallel simulation, (c) managed the equations of motion of all entities in the simulation for remote applications, (d) provided rollforward and rollback state management, (e) supported data logging for later playback, visualization, and analysis, and (f) provided fault tolerant mechanisms to handle external systems unexpectedly disconnecting from the simulation.

1996-2000: Missile Defense and Wargame 2000

As publish and subscribe simulation technologies, such as those articulated by the High Level Architecture (HLA), matured in the late 1990's, a next generation EMF was

[2] The EMF does not currently address multi-level security. This could be a very interesting topic for future research.

[3] The actual implementation was never given a formal name.

[4] An extremely simple event-processing engine was developed to manage rollbackable/rollforwardable internal event processing. Equations of motion and a few other attributes were specifically managed in a formal way.

[5] A nation-wide distributed missile defense simulation was demonstrated in 1993. The core optimistic simulation executed on several locally interconnected super computers at the Joint National Test Facility in Colorado Springs. A battle planner executed remotely on large supercomputer at the Los Alamos National Laboratory (LANL) in New Mexico. A flight simulator with real-time 3D terrain rendering executed at the Jet Propulsion Laboratory in California to find and destroy Transporter Erector Launchers (TEL) based on simulated tracks. Full visualization was performed at the Naval Research Laboratory (NRL) in Washington DC.

developed for SPEEDES that was based on Object Proxies.[6] This new EMF, known as the Object Proxy State Manager (OPSM), provided a much better foundation for supporting logical-time synchronization and external state management with an executing SPEEDES simulation. Object Proxies encapsulated state attributes into an object that could be published and then discovered by subscribing entities. Attribute update and reflection techniques were automated through cleverly devised operator overloading techniques. The Object Proxy subscription capabilities were extended into the OPSM, along with rollback and rollforward support. This provided a generic mechanism for external applications to obtain the state of the simulation at any point in time.[7]

2000-2004: Joint Simulation System (JSIMS)

SPEEDES became the Common Component Simulation Engine (CCSE) for the Joint Simulation System (JSIMS) in 2000. With hundreds of new model developers spanning multiple agencies, industrial organizations, the joint armed forces, and intelligence communities, it became clear that a more capable publish and subscribe framework was needed to meet JSIMS requirements. A more flexible and scalable framework based on Federation Objects, Hierarchical Grids, and internal Object Persistence was developed to meet the new demanding requirements posed by JSIMS.[8] A new EMF system was developed to support external tools such as the Model Diagnostic Driver Interface (MDDI) that was heavily used for two-way monitoring and controlling battlefield entities.

2001-Present: Open Unified Technical Framework

Development of the WarpIV Kernel[9] began in early 2001 as the next-generation replacement for SPEEDES. Based on three open system architectures,[10] the WarpIV Kernel evolved over the next decade into what is now the core infrastructure of the Open Unified Technical Framework (OpenUTF). The core technology layers of the OpenUTF, including the new EMF, are shown in Figure 2.

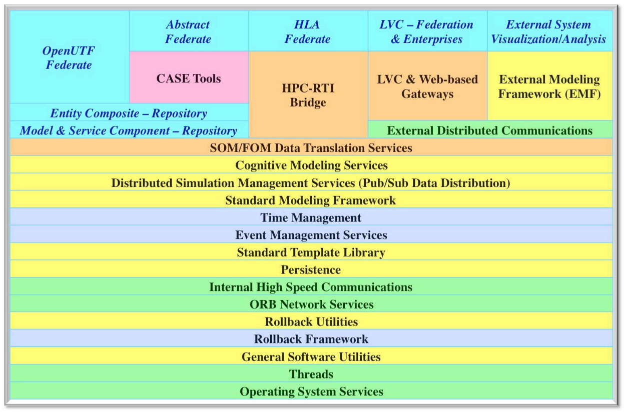
OpenUTF Federate
Abstract Federate
HLA Federate
LVC – Federation & Enterprises
External System Visualization/Analysis
CASE Tools
HPC-RTI Bridge
LVC & Web-based Gateways
External Modeling Framework (EMF)
Entity Composite – Repository
Model & Service Component – Repository
External Distributed Communications
SOM/FOM Data Translation Services
Cognitive Modeling Services
Distributed Simulation Management Services (Pub/Sub Data Distribution)
Standard Modeling Framework
Time Management
Event Management Services
Standard Template Library
Persistence
Internal High Speed Communications
ORB Network Services
Rollback Utilities
Rollback Framework
General Software Utilities
Threads
Operating System Services

Figure 2: Technology layers of the OpenUTF.

Leveraging lessons learned from JSIMS, a new, more capable, and easier-to-use publish and subscribe framework was developed. Built on the OpenUTF Object Request Broker (OpenUTF-ORB) client/server distributed communications framework, a new network-based EMF was developed to support optimized communications, robust time management, two-way event processing, and the extended Federation Object capabilities.

Besides providing much-optimized internal state and rollback management techniques,[11] the new EMF now offers support for the full OpenUTF modeling constructs within its internal event-processing infrastructure. This means that in addition to providing logical time and publish and subscribe state management, the EMF can also be used to host general-purpose sequential simulation applications that remotely connect and interact with an

[6] A simple HLA interface was developed to the 1996-2000 EMF to demonstrate how external HLA simulations could directly interface with SPEEDES. The first HLA High Performance Computing Run Time Infrastructure (HPC-RTI) prototype was based on this EMF. Later versions of the HPC-RTI built for SPEEDES and the WarpIV Kernel provided native interfaces where federates executed as nodes within the parallel simulation execution.

[7] Like its predecessor implementation, the 1996-2000 EMF offered a very simple event-processing engine. However, now state management was more formalized with object proxies.

[8] The 2000-2004 EMF coupled a very simple event processing engine with a much more elaborate Federation Object framework for managing state information.

[9] The WarpIV Kernel implementation of the OpenUTF core infrastructure contains more than 400,000 lines of C++, Java, and Python code. It executes on all mainstream platforms (Linux, Mac OS X, Windows, Solaris, HP-UX, etc.) and compilers. The WarpIV Kernel is provided freely with source code to U.S. organizations for non-commercial use.

[10] The OpenUTF is comprised of three architectures that are being investigated by the Simulation Interoperability Standards Organization (SISO) Parallel and Distributed Modeling & Simulation Standing Study Group (PDMS-SSG). These architectures are: (1) Open Modeling & Simulation Architecture (OpenMSA), which is a layered architecture that encapsulates each critical technology necessary for supporting parallel and distributed modeling and simulation applications, (2) Open System Architecture for Modeling & Simulation (OSAMS), which defines the set of programming interfaces for model developers, and (3) Open Cognitive Architecture Framework (OpenCAF), which provides a foundation for cognitive models representing intelligent human behaviors.

[11] Prior implementations of the EMF saved all event data such as messages and event structures when supporting rollback and rollforward capabilities. The OpenUTF EMF now cleans up all event data and only stores rollback items that are related to the event. This significantly reduces memory consumption within the EMF.

OpenUTF simulation executing optimistically on parallel and distributed computing resources. This is a significant improvement over prior EMF implementations.[12]

2.0 EMF Services

Code Segment 1 provides an example of a test program using the EMF. Normally, the EMF is constructed by the application in its main program. The EMF execution mode is then typically established.[13] Then the application subscribes to external events and/or Federation Objects that are produced by the simulation. Once these initialization steps are completed, applications are free to advance time and obtain state information about the federation objects that are discovered.[14]

Code Segment 1: Example application using the EMF.

```
// Example application

#include "WpEmf.H"
#include "WpExecute.H"

#define START_TIME 0.0
#define END_TIME 3600.0
#define LOOKAHEAD 10.0

int main(int argc, char **argv) {

// Construct the Emf

  WpEmf *emf = WpCreateEmf(argc, argv);

// Set the EMF to live mode and generate a
// log file. The log file name is optional
// and if left out, no log file is
// generated. Note that the EMF is set to
// playback mode by using the interface
// emf->SetPlayback("LogFile");

  emf->SetLive(
    START_TIME,
    LOOKAHEAD,
    "LogFile"
  );

// Subscribe to events and Federation
// Objects. In this example, "Fo" is the
// base class of all federation objects
// which means that the EMF will subscribe
// to all objects. Multiple calls to
// subscribe are permitted. The EMF can
// subscribe to all events using the
// call emf->SubscribeAllEvents();

  emf->SubscribeFo("Fo");
  emf->SubscribeEvent("AppleEvent");
  emf->SubscribeEvent("OrangeEvent");

// OpenUTF simulations can have initial
// named pauses to allow one or more
// external applications to launch before
// advancing time. Pauses are removed by
// making a call to resume from a pause.

  emf->Resume("StartUp");

// Now begin advancing time in a manner
// that paces with the simulation. The
// lookahead value passed in the SetLive
// method ensures that the EMF never lags
// behind the simulation by more than the
// lookahead value.

  double time = START_TIME;
  while (time < END_TIME) {

// Loop over all of the Federation Objects
// and obtain relevant state information.

    WpFo *fo = WP_EMF_FO_MGR.
      GetFirstRemoteFo("Aircraft");

    while (fo != NULL) {

// do something with the Federation Object
// such as obtain its position and
// orientation to draw the screen. Then
// get the next Federation Object.

      fo = EMF_FO_MGR.GetNextRemoteFo();
    }

// Advance time without blocking. Note that
// applications can provide an optional
// time argument to limit how far the Emf
// advances time within one non-blocking
// iteration.

    time = emf->AdvanceTime();
  }

// Now for fun, go backward and forward in
// time. The internal state of the Emf
// will reflect the state of the simulation
// at the requested times.

  emf ->RequestTime(100.0);
  emf ->RequestTime(500.0);
  emf ->RequestTime(1000.0);
  emf ->RequestTime(200.0);
```

[12] The new EMF supports the full OpenUTF modeling framework, which provides a great way for legacy applications to migrate models into the OpenUTF.

[13] Examples of execution modes are Live (with or without generating a log file) and Playback.

[14] The EMF is currently single-threaded, which means that live interactive applications involving humans in the loop must be careful to advance time in a manner that doesn't block inside the EMF. The *advance time* method is a non-blocking call that advances time as far as it can (or to the requested time if supplied as an argument) within a single iteration. Blocking during *time requests* is not an issue when operating the EMF in Playback mode.

```
// Here is an example of the Emf looping
// in 1 second simulated time steps from
// the start time to the end time of the
// simulation.

  for (time = START_TIME;
       time < END_TIME,
       time++) {

// Request the desired time.

    emf->RequestTime(time);

// Get state information about each
// simulated entity and then update the
// display.

  }

// Before exiting the application, destroy
// the EMF. Then, return a clean error code
// from the main program.

  WpDestroyEmf(emf);
  return 0;
}
```

Applications can directly schedule events for any of the objects that are modeled within the EMF using the full capabilities of the OpenUTF standard modeling framework. To maintain causality, applications must never schedule events with time tags that are in the past of their previous largest time request.

In addition, the application can schedule simple events[15] for any object residing within the OpenUTF parallel simulation. The time value must never be less than the previous largest time request plus lookahead. The following EMF methods are supported.

```
void WpEmf::ScheduleEvent(
  double time,
  const char *eventName
);

void WpEmf::ScheduleEventClass(
  double time,
  const char *eventName,
  void *data,
  int dataBytes
);

void WpEmf::ScheduleEventData(
  double time,
  const char *eventName,
  void *data,
  int dataBytes
);

void WpEmf::ScheduleEventRtc(
  double time,
  const char *eventName,
  WpRunTimeClass *rtc
);

void WpEmf::ScheduleEventRtc(
  double time,
  const char *eventName,
  WpRunTimeClass &rtc
);
```

Similar interfaces are provided for EMF applications to schedule events for the simulation as tightly in time as possible. These interfaces simply leave out the time parameter in the event-scheduling interfaces. The EMF automatically schedules the event with a time value set to the previous largest time request.

```
void WpEmf::ScheduleEvent(
  const char *eventName
);

void WpEmf::ScheduleEventClass(
  const char *eventName,
  void *data,
  int dataBytes
);

void WpEmf::ScheduleEventData(
  const char *eventName,
  void *data,
  int dataBytes
);

void WpEmf::ScheduleEventRtc(
  const char *eventName,
  WpRunTimeClass *rtc
);

void WpEmf::ScheduleEventRtc(
  const char *eventName,
  WpRunTimeClass &rtc
);
```

The default configuration of the EMF is to save rollback information for each event that is internally processed to support forward and backward time requests. For external EMF applications that do not require this capability (i.e., external systems that need to connect to the simulation and interact during execution), a method is provided to disable rollbacks. This method slightly reduces processing overheads, but can significantly reduce memory

[15] Simple events use a string argument to define the name of the event and come in four flavors: (a) no arguments, (b) a flat fixed-size class argument, (c) a variable-length buffer, and (d) arbitrary user data stored in a run-time class that provides name-value pairs. Event-handling methods are provided by the application as arbitrary methods on arbitrary objects. They are dynamically registered and unregistered during the execution of the simulation. Similarly, subscribe and unsubscribe methods within the OpenUTF provide a scalable mechanism for simple events to automatically be routed to the right subscribing objects.

consumption for large simulations with frequent attribute updates.

```
void WpEmf::DisableRollback();
```

Three additional methods are provided to pause, resume, and kill the simulation. Pauses are named by remote applications to support pause and resume requests from multiple sources. The EMF automatically appends socket file descriptors to the pause names to ensure uniqueness between distributed systems.

3.0 Technology of the EMF

The EMF provides robust and scalable technical solutions to several challenging distributed synchronization issues. Some of the more interesting technical issues and their solutions are discussed in this section.

– Synchronizing Messages with Time Updates –

The first issue involves how to coordinate time updates with time-tagged messages[16] that are flowing from a parallel optimistic simulation executing within the OpenUTF to one or more connected EMFs.

The solution is straightforward. First, a message is only released when the event that generated the message is committed. The simulation client that resides within each node of the OpenUTF automates this. Only valid messages are released to external systems. Messages are routed through the Network Server to EMFs based on subscriptions. Messages potentially flow from each node through the Network Server to the external systems where they are internally scheduled as EMF events as they are received. The EMF can only safely process events up to the time granted by the OpenUTF simulation. The EMF must never process events out of order. So events scheduled by various OpenUTF simulation nodes are received in arbitrary order by the EMF. Received messages must be coordinated with time update messages in a manner that guarantees no further messages will be received with time tags less than the time update. The solution is shown in Code Segment 2.

Code Segment 2: Actual code within the OpenUTF to synchronize time-tagged messages that were generated during event processing for external systems with time updates. Note that these steps are only performed if the simulation is configured at run time to support interactions with external systems.

```
// If the simulation client on this node
// has sent any messages during the current
// GVT cycle, reset the number and then
// perform a flush operation with the
// Network Server.

if (0 < SIM_CLIENT.GetNumMessagesSent()) {
  SIM_CLIENT.ResetNumMessagesSent();
  SIM_CLIENT.Flush();
}

// Synchronize with all nodes to guarantee
// that external time is not updated until
// all messages from each node have been
// received by the Network Server.

WpBarrierSync();
SIM_CLIENT.UpdateExtTime();
```

During each Global Virtual Time (GVT) event-processing cycle, each node keeps track of how many external messages are sent. Before sending the time update message to the Network Server, each node that sent at least one messages makes a two-way[17] flush call to the Network Server. This ensures that all messages sent by the node have been received and processed by the Network Server. Then, each node performs a high-speed barrier sync to ensure that all messages have been received and processed by the Network Server, where they are forwarded to external systems as appropriate. Then, the simulation client sends the time update message to the Network Server if it is node zero, which is then forwarded to the external systems. This ensures correct behavior because all messages were sent, received, and processed prior to the time update message, and because the Network Server processes and forwards messages in the order that they were received.

– Use of Lookahead –

Lookahead is required for maintaining worst-case lag between external applications and the OpenUTF simulation. Lookahead applies only to events being scheduled from the EMF to the OpenUTF simulation. No lookahead is required for events that are scheduled from the OpenUTF to the EMF. Each external application can have its own lookahead value. In addition, EMFs can change their lookahead value during execution if necessary. However, external applications must never schedule events tighter in time than any previously established time barrier[18] for the OpenUTF. If this ever

[16] Time tagged messages represent all events coming from the simulation to external systems using the EMF. Event types can be simple events, Federation Object discovery or removal, and attribute updates.

[17] Two-way message services within the OpenUTF-ORB blocks the application until the return message, potentially containing the results of the service, is received. In the case of the Flush service, no data is returned. Flush simply ensures that all messages sent by the client have been received and processed by the server.

[18] Time barriers limit GVT from advancing beyond the time value specified in the barrier. Time barriers are used for supporting pause and resume capabilities, throttling GVT to

occurs, the scheduled event time is automatically adjusted to be the barrier time plus lookahead.

Large values of lookahead promote more concurrent and scalable operation, but at the expense of limiting how tightly in simulated time external systems can interact with the OpenUTF. The current approach suffers from time creeping when lookahead values become very small. This issue was successfully addressed in earlier EMF implementations and will eventually be remedied in the OpenUTF implementation. The solution to the problem is to allow time to jump when there are no events scheduled between external systems and the OpenUTF. This technique can only be used for logical execution modes. Real time interactive simulations must always support spontaneous events that can be generated without warning.

– Fault Tolerance and Time Barriers –

Time barriers provide a formal way for external applications to limit Global Virtual Time (GVT) in the OpenUTF. Time barrier messages are sent to node zero of the OpenUTF from each EMF as they internally advance their logical time. Time barriers limit GVT from advancing beyond the granted time plus lookahead.

An important issue is how to handle situations when the external system unexpectedly disconnects[19] from the OpenUTF without cleanly removing any of its time barriers. If not handled correctly, the simulation would hopelessly be stuck from advancing beyond the time of the barrier. Fortunately, the Network Server is notified when an external system unexpectedly disconnects its socket connection. The Network Server keeps track of all time barriers and smartly removes them from node zero of the OpenUTF when external systems disconnect.

– Internal Management of Logical Time –

The EMF supports the ability to process internal events, roll them back in time when earlier times are requested, and then roll those events forward as necessary when forward time advances are requested. Supporting all of this produces some interesting challenges in managing the set of unprocessed pending events with those events that have been processed, but potentially rolled back.

the wall clock in support of scaled time operation, and for supporting robust time management between external systems and the OpenUTF. The EMF establishes time barriers for the OpenUTF based on its granted time plus the lookahead value.

[19] External systems can unexpectedly disconnect due to software crashes, hardware failures, network problems, power outages, or improper shutdown of an external system as it completes its task.

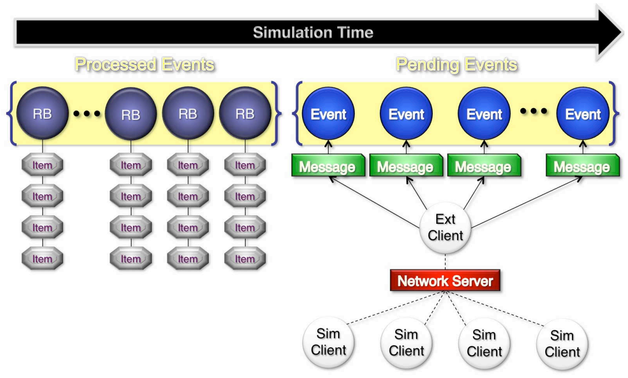

Figure 3: Event management within the EMF. This figure shows messages generated by the OpenUTF that are sent through the simulation client that resides on each node. Messages flow through the Network Server to the External Client that resides within the EMF. These messages are queued up as unprocessed events. As events are processed, their rollback information is saved in a queue for rollback and rollforward support. Moving forward and backward in time requires the EMF to manage all of this in a robust manner.

For example, suppose the application first requests to advance to time 1000 and subsequently requests going back to time 500. All events processed between time 500 and 1000 must be rolled back in the reverse order that they were processed. Then, suppose the external application requests to advance time to 1200. The EMF must roll forward those events that were previously rolled back from time 500 to time 1000 in forward order and then process any unprocessed events up to time 1200. Of course, all new event processing must be performed using lookahead and time barriers that coordinate with the OpenUTF simulation because there may be new events received by the EMF from the simulation during event processing between time 1000 and 1200.

The EMF provides four private methods that are not visible to applications to handle this.

```
void WpEmf::GoToTime(double time);
void WpEmf::ProcessUpToTime(double time);
void WpEmf::RollforwardToTime(double time);
void WpEmf:: RollbackToTime (double time);
```

The GoToTime method compares the requested time argument to the CurrentTime value that is maintained by the EMF as time requests are made. If the requested time is less than the current time, then the RollbackToTime method is called. If the requested time is greater than the current time, the RollforwdToTime method is called, followed by a call to the ProcessUpToTime method.

Two easy-to-use time-related EMF services are provided to external applications.

```
double WpEmf::AdvanceTime(
  double time=WP_INFINITY
);

double WpEmf::RequestTime(
  double time,
  int maxIterations=1000000000
);
```

The RequestTime service attempts to move to the requested time argument. It always succeeds when rolling back to earlier time values or when rolling forward to already-processed event times. When requesting time beyond prior grants, the AdvanceTime method tries to process events in coordination with the OpenUTF in time barrier cycles. If the maximum specified number of iterations is exceeded, the returned time is simply a best effort attempt. So, the granted time might not always be the same as the requested time.

The AdvanceTime method simply calls the RequestTime method using a maximum number of iterations of one. The AdvanceTime method should be used as opposed to RequestTime for interactive simulations that cannot afford to block while waiting for time to advance.

4.0 EMF and Next-generation Standards

The Parallel and Distributed Modeling & Simulation Standing Study Group (PDMS-SSG) operates within the Simulation Interoperability Standards Organization (SISO) and provides a forum for investigating open architecture standards, such as OpenMSA, OSAMS, and OpenCAF. The EMF techniques described in this paper heavily rely on the layered OpenMSA technologies. The EMF is a necessary part of the set of OpenUTF architecture standards that are coordinated by the OpenUTF Users Group and broader participating communities within SISO.

5.0 Summary and Conclusions

This paper first provided an introduction to the EMF by describing its high-level capabilities and then discussing historical implementations that date back to 1990. Each EMF implementation provided significant enhancements over previous efforts. The current EMF in the OpenUTF is a fourth generation system that benefitted from lessons learned over the past twenty years.

This paper then provided a high-level overview of the services provided by the EMF with an example to demonstrate its programming interfaces.

Finally, this paper discussed some of the key technical solutions used by the EMF to support (a) robust time management with optimistic simulations, (b) the role of lookahead to provide concurrency between external systems and the OpenUTF, (c) internal time management techniques for integrating rollback, rollforward, and event processing with time in the OpenUTF, and (d) fault tolerance to ensure proper time management if external systems using the EMF fail.

The EMF provides an extremely powerful, yet simple, interface to integrate remote systems across arbitrary networks with OpenUTF applications executing in parallel. The EMF supports numerous types of external applications including: (a) graphical visualization, (b) distributed analysis, (c) hardware-in-the-loop, (d) training systems, (e) support for legacy systems, (f) integration with real-world applications, and (g) integration with test articles using the OpenUTF as a robust testbed to support verification, validation, and accreditation efforts.

Finally, the SISO PDMS-SSG will likely investigate the EMF for future standardization within the OpenUTF.

6.0 Future Development

The EMF does not currently support publishing of its own Federation Objects that are discovered by the OpenUTF simulation, nor does it support the ability to modify the attributes of remote Federation Objects that were published by entities within the OpenUTF simulation. These capabilities are straightforward to implement and will be developed at a future date when user requirements demand this feature.

An HLA interface may eventually be developed for the EMF to support direct integration of OpenUTF applications with legacy HLA federates.

7.0 Acknowledgements

The authors of this paper would like to thank Cynthia Williams and the Missile Defense Agency (MDA) for sponsoring the work described in this paper within the scope of a current Phase II SBIR effort with WarpIV Technologies, Inc. The EMF will provide remote interfaces for several OpenUTF visualization and analysis capabilities that will be developed on this effort.

The authors of this paper would like to thank Chris Gaughan, Deputy Technology Program Manager of the U.S. Army Research, Development and Engineering Command (RDECOM) Modeling Architecture for Technology, Research & EXperimentation (MATREX) program. Mr. Gaughan is the Vice Chair of the PDMS-SSG and is actively involved in bringing multicore computing to Army Force Modeling Simulation (FMS) and Chemical and Biological Defense (CBD) programs.

The authors of this paper would like to thank Bob Pritchard, Program Manager of the Joint Virtual Network Centric Warfare (JVNCW) program at SPAWAR

Systems Center Pacific, for his continued support of the OpenUTF. Bob Pritchard is a strong advocate of the PDMS-SSG and is a proponent of open source software methodologies.

8.0 References

1. Blank Gary, Steinman Jeff, Shupe Scott, Wallace Jeff, 2000. "Design and Implementation of the HPC-RTI for the High Level Architecture in SPEEDES 0.81." In the proceedings of the *2000 Spring Simulation Interoperability Workshop*. Paper 00S-SIW-153.
2. Clark Joe, Capella Sebastian, Bailey Chris, Steinman Jeff and Peterson Larry, 2002. "The Development of an HLA Compliant High Performance Computing Run-time Infrastructure" In proceedings of the *2002 Spring Simulation Interoperability Workshop, Paper 02S-SIW-016.*
3. Lammers Craig, Valinski Maria, Steinman Jeff, 2009. "Multiplatform Support for the OpenMSA/OSAMS Reference Implementation." In proceedings of the Spring 2009 Simulation Interoperability Workshop (SIW).
4. Parallel and Distributed Modeling & Simulation Standing Study Group (PDMS-SSG) Terms Of Reference (TOR).
5. Parallel and Distributed Modeling & Simulation Standing Study Group (PDMS-SSG), 2009. "Parallel and Distributed Modeling & Simulation Standing Study Group (PDMS-SSG) DRAFT REPORT Volume 1: PDMS Technology." Submitted to the SISO Standards Activities Committee (SAC), 14 April 2010.
6. Steinman Jeff, 1991. "SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation." In proceedings of *Advances in Parallel and Distributed Simulation*. Pages 95-103.
7. Steinman Jeff, 1991. "Interactive SPEEDES." In proceedings of the *24th Annual Simulation Symposium*. Pages 149-158.
8. Steinman Jeff, 1993. "Incremental State Saving in SPEEDES Using C++." In proceedings of the *1993 Winter Simulation Conference*. Pages 687-696.
9. Steinman Jeff, 1993. "Synchronization of Parallel and Distributed Interactive Military Simulations Using SPEEDES." In proceedings of the *1993 Summer Computer Simulation Conference*. Pages 701-710.
10. Simulation Interoperability Standards Organization (SISO), http://www.sisostds.org.
11. Steinman Jeff, 1998. "Time Managed Object Proxies in SPEEDES." In the proceedings of the *Object-Oriented Simulation Conference* (OOS'98), pages 59-65.
12. Steinman Jeff, et. al., 1999. "Design of the HPC RTI for the High-Level Architecture", In proceedings of the *1999 Fall Simulation Interoperability Workshop*, Paper 99F-SIW-071.
13. Steinman Jeff, et. al., 1999. "The SPEEDES-Based Run-Time Infrastructure for the High-level Architecture on High-Performance Computers." In proceedings of the *High Performance Computing 1999 Conference, Grand Challenges in Computer Simulation*. Pages 255-266.
14. Steinman Jeff and Hardy Doug, 2004. "Evolution of the Standard Simulation Architecture." In proceedings of the *Command and Control Research Technology Symposium*, paper 067.
15. Steinman Jeff, 2005, "The WarpIV Simulation Kernel." In proceedings of the *2005 Principles of Advanced and Distributed Simulation (PADS) conference.*
16. Steinman Jeff, Park Jennifer, Walters Wally, and Delane Nathan, 2007. "A Proposed Open System Architecture for Modeling and Simulation." In proceedings of the *Fall 2007 Simulation Interoperability Workshop*, 07F-SIW-044.
17. Steinman Jeff and Lammers Craig, 2007, "WarpIV Object Request Broker User's Guide: Version 1.5." Copyright © 2007, WarpIV Technologies, Inc.
18. Steinman Jeff, Lammers Craig, Valinski Maria, 2008. "A Unified Technical Framework for Net-centric Systems of Systems, Test and Evaluation, Training, Modeling and Simulation, and Beyond…" In proceedings of the Fall 2008 Simulation Interoperability Workshop (SIW). Paper 08F-SIW-041.
19. Steinman Jeff, Lammers Craig, Valinski Maria, and Steinman Wendy, 2009. "Open Unified Technical Framework Volume 1: OSAMS, User's Guide for Model Developers." WarpIV Technologies, Inc.
20. Steinman Jeff, Lammers Craig, and Valinski Maria, 2009. "A Proposed Open Cognitive Architecture Framework (OpenCAF)." In the proceedings of the 2009 Winter Simulation Conference.
21. Steinman Jeff, Lammers Craig, Valinski Maria, 2009. "Composability Requirements for the Open Unified Technical Framework." In proceedings of the Fall 2009 Simulation Interoperability Workshop, paper 09F-SIW-022.
22. Steinman Jeff, Lammers Craig, Valinski Maria, Steinman Wendy, 2009. "Migrating Software to the Open Unified Technical Framework." In proceedings of the Spring 2010 Simulation Interoperability Workshop, paper 10S-SIW-052.
23. Steinman Jeff, Lammers Craig, Valinski Maria, and Steinman Wendy, 2010. "Scalable Publish and Subscribe Data Distribution." In proceedings of the *Fall 2010 Simulation Interoperability Workshop*, 10F-SIW-027.
24. Tung Yu-Wen and Steinman Jeff, 1993. "Interactive Graphics for the Parallel and Distributed Computing Simulation." In proceedings of the *1993 Summer Computer Simulation Conference*. Pages 695-700.
25. WarpIV Technologies, Inc. 2009. "Open Unified Technical Framework (OpenUTF) Volume 1: OSAMS User's Guide for Model Developers."

9.0 Author Biographies

JEFFREY S. STEINMAN, President and CEO of WarpIV Technologies, Inc. received his Ph.D. in High Energy Physics from UCLA in 1988 where he studied the quark structure function of high-energy virtual photons at the Stanford Linear Accelerator Center. Dr. Steinman was the creator of the Synchronous Parallel Environment for

Emulation and Discrete Event Simulation (SPEEDES), has published more than 70 papers and articles in the field of high performance computing, has five U.S. patents in high performance M&S technology, is a frequent invited speaker at conferences, seminars, and forums, and is the principle designer/developer of the WarpIV Kernel Reference Implementation of the OpenMSA, OSAMS, and OpenCAF architectures that make up the Open Unified Technical Framework (OpenUTF). Dr. Steinman is currently the Chair of the Parallel and Distributed Modeling & Simulation Standing Study Group (PDMS-SSG) that is governed by the Simulation Interoperability Standards Organization (SISO).

CRAIG N. LAMMERS, Vice President of Applied Research and Development at WarpIV Technologies, Inc. is the program manager for an effort with AFRL, Rome Labs conducting new research and development in parallel and distributed simulation to support formal estimation and prediction of complex systems using the OpenUTF Kernel. He is also the lead engineer for a new research effort with the Missile Defense Agency applying five-dimensional simulation to support the analysis of interceptor engagement strategies. His professional interests are in the areas of M&S, artificial intelligence, signal processing, user interface design, and music. Mr. Lammers received his B.S. degree in Industrial and Systems Engineering from the University of Michigan, Dearborn in 2001, where he graduated with high honors.

MARIA E. VALINSKI, Vice President of Software Engineering at WarpIV Technologies, Inc., has provided modeling, simulation, and integration technical support for numerous programs and organizations. Maria Valinski is the lead software engineer for the Joint Virtual Network Centric Warfare (JVNCW) program that models wireless communications on supercomputers using the OpenUTF reference implementation. Maria Valinski is currently providing Subject Matter Expert (SME) support for OpenUTF users at the Space and Naval Warfare (SPAWAR) Systems Center Pacific (SSC-PAC). Ms. Valinski received her B.S. degree in Computer Science from East Stroudsburg University in 1997.

WENDY L. STEINMAN, Software Engineer at WarpIV Technologies, Inc. received her B.S. degree in Cognitive Science specializing in Neuroscience with a minor in Biology from UCSD in 2009. She is currently supporting several initiatives involving the OpenUTF. This includes developing data registry services, implementing two and three dimensional grid decomposition strategies for simulation objects, exploring hybrid computation techniques, conducting performance benchmarks, comparing the internal OpenUTF High Speed Communications (HSC) library with the standard Message Passing Interface (MPI), analyzing M&S data provided by the Chemical, Biological Radiological and Nuclear Defense (CBRND) Data Backbone, assisting in writing the OpenUTF User's Guide, and editing the Parallel and Distributed Force Modeling & Simulation Technical Report for the PDMS-SSG.